Apontamentos sobre Bases de Dados

Base de Dados (definição) – Colecção de dados organizados de forma a serem facilmente mantidos, actualizados e pesquisados. – Uma base de dados é uma colecção, com descrição sobre ela própria, de registos integrados.

História e Evolução das BD – Os primeiros sistemas de informação eram baseados em conjuntos de ficheiros que guardavam conjuntos de registos.(***file processing systems***)

As principais limitações deste sistema eram :
1 Os dados ficavam separados e isolados - obter dados de mais que um ficheiro, em simultâneo, era uma tarefa complicada. Tinham que ser as aplicações a fazer essas tarefas.
2 Duplicação de dados - a mesma informação existe em mais que um ficheiro tornando complicado os processos de manutenção e normalmente levando a problemas de integridade dos dados.
3 Os programas que acediam aos dados eram dependentes do formato dos ficheiros, se os formatos mudassem tinham que ser alterados esses programas.
4 Ficheiros muitas vezes incompatíveis uns com os outros.
5 Difícil representar os dados na perspectiva do utilizador.

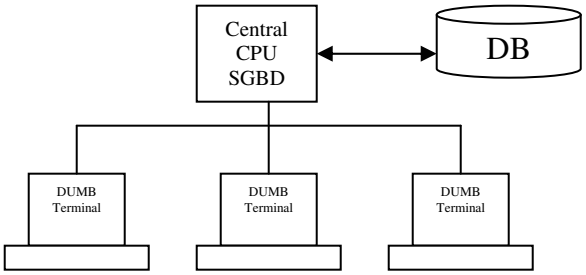
As tecnologia de bases de dados foram criadas numa tentativa de superar estas limitações :
1 Os dados são todos guardados na mesma base de dados. Quando são necessários dados de entidades diferentes, o programa apenas especifica como é que os dados são combinados mas é o gestor da base de dados que efectua as operações, necessárias, para dar a resposta.
2 A duplicação de dados está limitada. Os dados são guardados apenas num sítio e portanto os problemas de integridade são limitados.
3 Os programas não acedem directamente aos dados, apenas fazem pedidos ao SGBD e é este que fornece os dados. Os programas ficam menos dependentes dos formatos dos ficheiros.
4 Não há ficheiros
5 É relativamente fácil representar a estrutura de dados

O Modelo Relacional
Em 1970 E.F. Codd publicou um paper onde usava conceitos matemáticos para resolver o problema de como guardar grandes quantidades de dados.
Este paper levou mais tarde ao aparecimento do modelo relacional.
No modelo relacional os dados estão organizados em tabelas com linhas e colunas.
Devido à maneira como os dados estão organizados é minimizado o problema de duplicação de dados.
Este modelo não teve, no inicio, muito sucesso devido aos enormes recursos que necessitava, o que o tornava muito lento para o hardware existente.
Na actualidade este modelo de organização é o mais utilizado pela maioria dos sistemas de gestão de bases de dados.

Partilha de dados na empresa

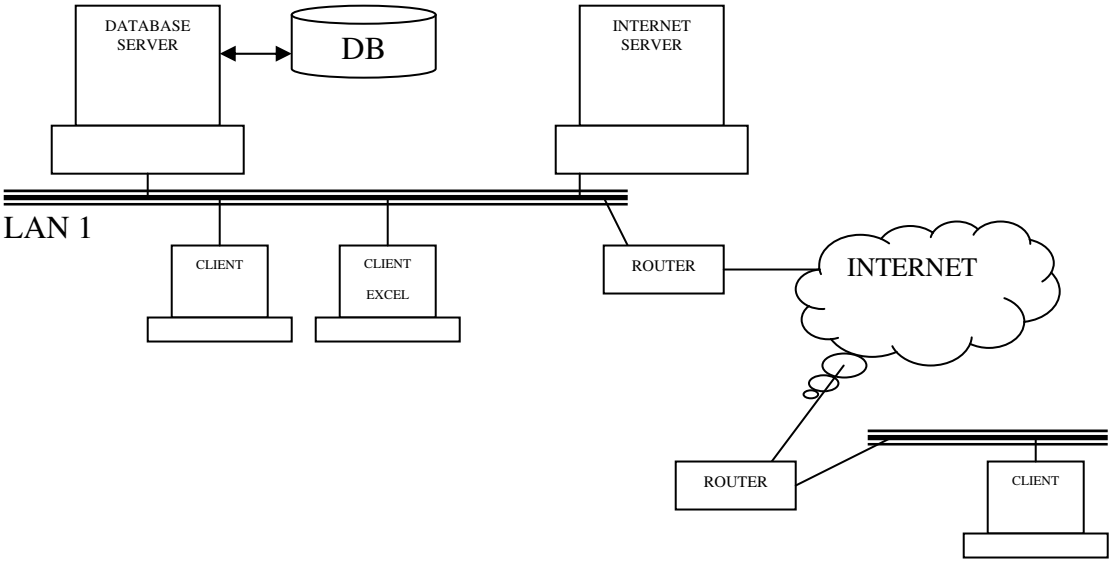
Sistemas de Teleprocessamento

É o método clássico para suportar uma base de dados multiutilizador. Só existe um CPU e todo o processamento é feito por este único computador. Os utilizadores usam terminais estúpidos para transmitir as mensagens ao computador central, este faz todo o processamento necessário e o resultado e transmitido ao terminal que fez o pedido. Como os terminais são estúpidos, até a formatação do écran do terminal, tem que ser feita pelo CPU central através das linhas de comunicação. Para que não haja muito trafego nas linhas os terminais são, geralmente, apenas de caracteres.

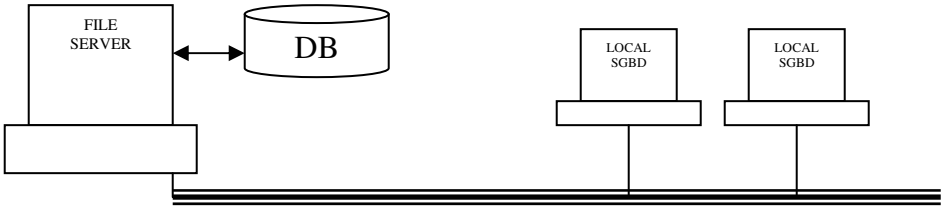


Sistemas Cliente Servidor

Este sistema implica uma rede de computadores. Alguns destes computadores processam as aplicações e são chamados *Clientes,* outro processa a Base de Dados e é chamado *Servidor*. As aplicações cliente são responsáveis por todo o interface, fazem o pedido ao servidor que processa esse pedido e devolve o resultado ao cliente.



Sistemas de partilha de ficheiros
Também implica uma rede de computadores. Existe um servidor de ficheiros e cada computador tem acesso a esses ficheiros. Cada computador tem também o seu próprio software de gestão da base de dados. Todos os pedidos e processamento da base de dados são efectuados localmente embora efectuados sobre ficheiros colocados no servidor de ficheiros. Muito mais trafego na rede do que no *cliente servidor* uma vez que as operações são feitas localmente sobre ficheiros colocados no servidor.

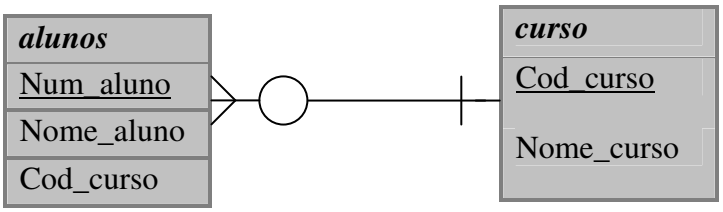


Sistemas distribuídos
Sistema em que a própria base de dado está distribuída. A Base de dados (ou parcelas dela estão guardadas em N computadores), alguns destes computadores processam as aplicações e a Base de Dados. Um outro processa apenas a Base de Dados.

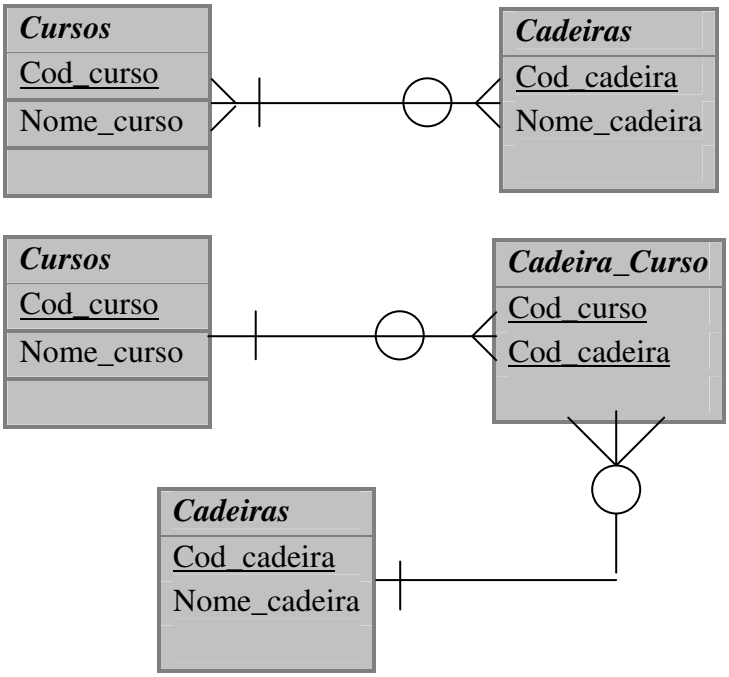
Modelos de dados

Estabelecer o modelo de dados é o processo de criar uma representação lógica da estrutura da base de dados. É a tarefa mais importante no desenvolvimento da base de dados. O modelo de dados deve representar, fielmente, a maneira como os utilizadores finais vem os dados.
O Modelo Entidade Relacionamento foi introduzido em 1976 por Peter Chen tendo sido, posteriormente, desenvolvido e modificado pelo próprio e por outros.
Elementos do Modelo Entidade Relacionamento :
Entidades – entidade é uma classe distinta de coisas que representam a realidade do utilizador e sobre as quais se sabe alguma coisa. Exp: professores, alunos, disciplinas....
Atributos – as entidades tem *atributos* ou *propriedades* que descrevem as características da entidade. Exp: numero de aluno, nome,...etc
Alguns atributos não só descrevem características da entidade como a permitem identificar uma única ocorrência da entidade. Chamamos chave (ou chave primária) a este especial atributo. Se a chave for constituída por mais que um atributo chama-se chave composta.
Exp : Numero de aluno na entidade *Aluno*.
Relacionamentos – Já vimos que uma entidade pode ser descrita pela sua chave primária e outros atributos que não fazem parte da chave. No entanto as entidades não podem ser vistas isoladamente pois na realidade podem existir relacionamentos entre algumas entidades. Exp: se existir uma entidade chamada Curso (com características que descrevem os vários cursos duma Universidade) haverá um relacionamento entre a entidade *aluno e curso* uma vez que o aluno frequenta um dado curso. Os relacionamento podem ser 1:1, 1 : many, many : many

***Diagramas Entidade Relacionamento* (DER)**



Exemplo de uma relação 1 : many
Um curso pode ter vários alunos, um aluno só anda num curso
Um aluno só pode ter um cod_curso para o qual exista já uma ocorrência na entidade *curso*. Um curso pode ou não ter alunos.
Relações many : many não podem ser implementadas directamente no modelo relacional senão à custa de uma entidade intermédia construindo 2 relações 1: many
Exp: Relação entre cursos e cadeiras – entidade intermédia curso_*cadeira*



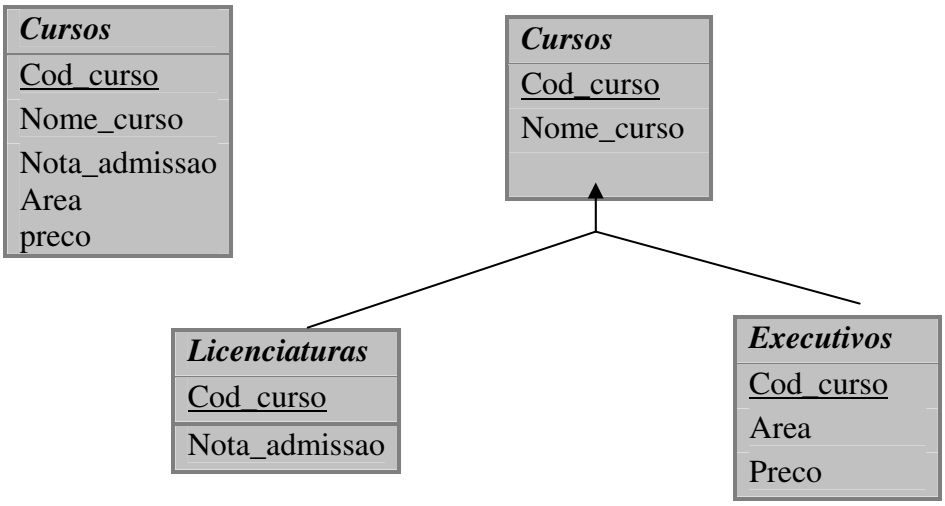
Exemplo de possíveis ocorrências destas entidades

Cursos	
Cod_curso	**Nome_curso**
001	Gestão
002	Economia

Cadeira_Curso	
Cod_curso	**Cod_cadeira**
001	023
002	023
001	027

Cadeiras	
Cod_cadeira	**Nome_cadeira**
023	Informática
027	TSII

Supertypes e SubTypes : O SuperType Cursos pode ter dois SubTypes que herdam as suas caracteristicas Cursos de Licenciaturas e Cursos de Executivos: Os Subtypes só podem existir se existir o SuperType. Executivos pode precisar de atributos tais como duração, preço do curso, area profissional a que se destina..etc e Licenciatura pode precisar de nota minima de admissão .



Bases de Dados Relacionais

Bases de dados relacionais são aquelas que estão organizadas segundo o modelo relacional. Este modelo é muito importante pois é utilizado pela maioria dos sistemas gestores de bases de dados.

O Modelo Relacional

Uma relação – é uma tabela de duas dimensões constituída por linhas e colunas: cada coluna representa dados referentes a um atributo e chamam-se ***campos***, linhas representam ocorrências dos vários atributos e chamam-se ***registos***. Para que uma tabela possa ser considerada uma relação é necessário que cada célula contenha um e um só valor, não há grupos repetidos, na mesma coluna os valores são do mesmo tipo, não há linhas iguais e a ordem das linhas e das colunas é irrelevante.

Dependência funcional – Existem numa tabela quando um ou mais atributos dependem de outros atributos. Isto é se para cada ocorrência do atributo A corresponder sempre o mesmo valor do atributo B, podemos dizer que B depende funcionalmente de A. Exp Codigo Postal e Localidade, podemos dizer que Localidade depende de Código Postal porque sempre que existir uma ocorrência do mesmo Código Postal teremos sempre o mesmo valor para Localidade. Como Codigo Postal determina Localidade chamamos a esse campo um campo ***determinante.***

Chaves – Chave primária

Alguns *campos* não só descrevem características da tabela como a permitem identificar uma única ocorrência de um registo. Se houver mais que um conjunto de campos nestas circunstancias chamamos a cada conjunto ***chave candidata.*** A chave candidata escolhida para chave tem o nome de ***chave primária***) Se a chave for constituída por mais que um campo chama-se chave composta.

Normalização

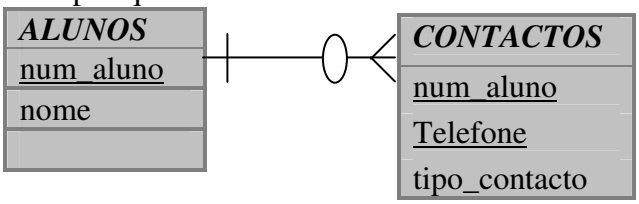
A Normalização minimiza a duplicação de dados de forma a garantir a integridade dos mesmos. É a base para remover indesejáveis *dependências funcionais* das tabelas.

1ª forma Normal – uma tabela que respeite as regras de uma relação está na 1ª forma normal (todos as repetições devem ser retiradas e colocadas numa nova tabela)

Exp:

ALUNOS
num_aluno
nome
tel1
tel2
tel3

Um aluno só poderia ter no máximo 3 contactos telefónicos, e se tiver menos que 3 há campos que ficam vazios



2ª forma Normal – uma tabela para estar na 2ª forma normal tem que já estar na 1ª e adicionalmente todos os campos que não fazem parte da chave devem depender de todos os componentes da chave. Ou seja não podem depender de apenas uma parcela da chave.(só tem sentido para tabelas com chaves compostas)

Consideremos a seguinte tabela correspondente a possíveis actividades desportivas que os alunos tenham :

NUM_ALUNO	ACTIVIDADE	PRECO
150196001	tenis	3000
150196001	golf	5000
150196002	tenis	3000
150196003	natação	2000
150196003	basquetebol	1500
150196004	basquetebol	1500
150196004	tenis	3000

A chave primária desta tabela é constituída por NUM_ALUNO + ACTIVIDADE uma vez que um aluno pode ter mais que uma actividade, e supondo que só nos referimos às actividades actuais dos alunos (sem histórico).

Como podemos ver, preço depende funcionalmente de actividade e não depende do aluno ou seja só depende de uma parte da chave primária.

Se apagarmos a linha 2 apagamos também o facto de que golfe custa 5000. Uma actividade só pode ser registada se houver um aluno inscrito nessa actividade. Esta tabela sofre de anomalias originadas pelo apagar de um registo, e problemas na inserção de novos registos.

O problema pode ser resolvido decompondo a tabela em duas; ACTIVIDADES e ACTIV_ALUNOS

ACTIV_ALUNOS

NUM_ALUNO	ACTIVIDADE
150196001	tenis
150196001	golf
150196002	tenis
150196003	natação
150196003	basquetebol
150196004	basquetebol
150196004	tenis

ACTIVIDADES

ACTIVIDADE	PRECO
tenis	3000
golf	5000
natação	2000
basquetebol	1500

Melhor ainda cada actividade deveria ter um código e na tabela ACTIV_ALUNO só deveria aparecer esse código....

3ª forma Normal – Para uma tabela estar na 3ª forma normal tem que já estar na 2ª e adicionalmente todos os campos que não fazem parte da chave só podem depender da chave. Ou seja não podem depender também de outros campos que não façam parte da chave.
Exp: Vamos supor que a universidade tinha várias residenciais onde os alunos podiam viver

num_aluno	edificio	preco
150196001	B1	40000
150196002	B2	50000
150196005	B1	40000

Como um aluno reside apenas num dos edifícios, ***num_aluno*** pode ser chave primária da tabela pelo que ela já se encontra na 2ª forma normal .No entanto não se encontra na 3ª forma normal uma vez que o campo ***preco*** depende do campo ***edificio*** que não faz parte da chave primária. Se tentarmos apagar a linha 2 perdemos também a informação do preço correspondente a B2. Também não é possível registar um novo edifício sem que haja pelo menos um aluno atribuído a esse edifício.O problema pode ser resolvido, tal como no exemplo anterior, decompondo a tabela em duas; **RESIDENCIAIS** e **RESIDENCIAIS_ALUNOS**

RESIDENCIAIS_ALUNOS

num_aluno	edificio
150196001	B1
150196002	B2
150196005	B1

RESIDENCIAIS

edificio	preco
B1	40000
B2	50000

Boyce-Codd Normal Form
Para uma tabela estar em BCNF é necessário que todos os determinantes sejam chaves candidatas.
No exemplo que se segue vamos supor que cada professor só dá uma cadeira. Temos pois como ***chaves candidatas*** (num_aluno, cadeira) ou (num_aluno,professor). Além disso cadeira depende funcionalmente de professor. *Professor* é pois um ***determinante***.

Se a ***chave primária*** for (num_aluno, cadeira) a tabela está na 3ª forma normal mas não se encontra em BCNF pois professor é ***determinante*** e não é por si só uma ***chave candidata***.
A tabela tem anomalias ao apagar registos (se for por exemplo apagado o último registo deixamos de saber que Guedes é professor de Fisica) e não podemos meter um professor novo numa cadeira até que um aluno tenha essa cadeira.

num_aluno	cadeira	professor
150196001	Matematica	Rebelo
150196002	Fisica	Lucas
150196005	Matematica	Rebelo
150196005	Fisica	Guedes

Mais uma vez a solução passa por decompor esta tabela em duas; (professor é ***chave primária*** da tabela professor-cadeiras)

num_aluno	professor
150196001	Rebelo
150196002	Lucas
150196005	Rebelo
150196005	Guedes

professor	cadeira
Rebelo	Matematica
Lucas	Fisica
Guedes	Fisica

4ª Forma Normal
Uma tabela está na 4ª forma normal se estiver em BCNF e, além disso, não existirem dependências multi-valor.
Numa tabela T(a,b,c) existem dependências multi-valor se ***a*** determinar valores múltiplos de ***b***, ***a*** determina múltiplos valores de ***c,*** e se ***b*** e ***c*** forem independentes um do outro.
Vamos supor uma tabela que mostra os relacionamentos entre alunos, disciplinas e desportos:

num_aluno	cadeira	desporto
150196001	Matematica	Tenis
150196001	Fisica	Tenis
150196001	Matematica	Natação
150196001	Fisica	Natação
150196005	Matematica	Basquetebol

A ***chave primária*** desta tabela tem que ser o conjunto dos 3 campos (num_aluno, cadeira, desporto). Para o mesmo aluno podem existir:
- várias ocorrências de cadeiras
- várias ocorrências de desportos.

Além disso, cadeiras e desportos são independentes um do outro. Assim tem que existir para cada aluno todas as combinações de cadeiras e desportos. Esta tabela contém, pois, multi-dependências. Este facto leva a anomalias de actualização. Se o aluno '150196001' deixar de praticar 'Tenis', não basta apagar um registo; é preciso apagar todos os que referem 'Tenis' como desporto para esse aluno. Tem que ser apagado um número de registos igual ao número de cadeiras que o aluno tem. Da mesma forma, se este mesmo aluno aderir a um novo desporto, obriga a que seja introduzido mais que um registo: tem que ser introduzido um registo por cada cadeira do aluno. Neste caso se ele passar a jogar 'Basquetebol' tem que ser introduzidos os registos [150196001,Matematica,Basquetebol] e [150196001,Fisica,Basquetebol].
Como de costume, a forma de resolver este problema é decompor esta tabela em duas outras (neste caso sem qualquer tipo de relacionamento entre elas); ***Alunos_Cadeiras*** e ***Alunos_Desportos***

num_aluno	cadeira
150196001	Matematica
150196001	Fisica
150196005	Matematica

num_aluno	desporto
150196001	Tenis
150196001	Natação
150196005	Basquetebol

Modelo Conceptual e modelo lógico

O modelo conceptual deve ser suficientemente geral para não depender das características particulares da Base de Dados onde vai ser implementado. Partindo do modelo conceptual é construído o modelo lógico onde serão feitas as modificações, necessárias, tendo em conta essas características. Poderá, também, ser efectuada algum tipo de desnormalização para garantir performance.

Fases importantes do Modelo Conceptual:

Definição das entidades
Definição das propriedades dessas entidades
Estabelecer os relacionamentos entre entidades
Normalizar as entidades

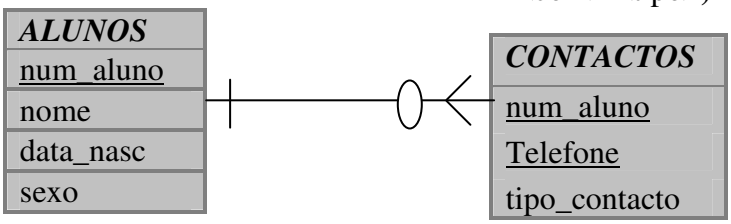
SQL – Structured Query Language

Principais tipos de dados :

CHAR(n) – string de comprimento fixo
VARCHAR(n) VARCHAR2(n) - string de comprimento variável com um máx de n chars. Varchar pode ter no max 255 chars...Varchar2...64k (65535)
NUMBER(n) ou NUMBER(n,d) – valor numérico constituído por n dígitos ou n dígitos dos quais d decimais (n pode tomar valores entre 1 e 38)
INTEGER - inteiro
FLOAT - número real
DATE – data e hora
RAW(n) LONG RAW(n) – para guardar binary data : exemplo bytes de uma imagem.RAW – pode guardar 255 bytes LONG RAW 64k (65535)
(em Access number corresponde a "double" e nao se podem definir o número de casas)

Criação de tabelas

CREATE TABLE <table_name> **(** <col1> <tipo>,
<col2> <tipo>,
<coln> <tipo>**)**



CREATE TABLE alunos (num_aluno number(9) not null,
nome varchar(40) not null,
data_nasc date ,
sexo char(1));

CREATE TABLE contactos(num_aluno number(9) not null,
telefone char(12) not null,
tipo_contacto number(3));

Alteração de tabelas

ALTER TABLE <table_name>
ADD <nome_coluna> <tipo>
ALTER TABLE <table_name>
MODIFY <nome_coluna> <tipo>
(em Access MODIFY é substituido por ALTER)
ALTER TABLE <table_name>
DROP <nome_coluna>

Exp: para alterar a tabela anterior de forma a o nome do aluno poder ter 50 chars

ALTER TABLE alunos MODIFY nome varchar(50)

Exp: adicionar a constraint ***primary key*** à tabela alunos
ALTER TABLE alunos ADD constraint pk_alunos
Primary key (num_aluno)
Exp: Adicionar uma check constraint para garantir que sexo só pode tomar os valores M ou F
ALTER TABLE alunos ADD constraint ck_sexo
Check (sexo in('M','F'))
(CHECK não existe em ACCESS)
Exp: criar a constraint ***foreign key*** entre as tabelas contactos e alunos – esta constraint vai garantir que não possam ser introduzidos contactos para alunos não existentes na tabela de contactos

ALTER TABLE contactos ADD constraint fk_contactos_alunos
Foreign key (num_aluno)
References alunos(num_aluno)

Apagar tabelas

DROP TABLE <table_name>

Exp: **DROP TABLE alunos**

Editar tabelas
.Introdução

INSERT INTO <nome_tabela> **(** <col1>,…<coln>**)**
VALUES (<Valor1>,…..,<Valorn>**)**

Exp : INSERT INTO alunos VALUES (150196001,'Carlos Pina','03-Dez-1973','M');
INSERT INTO alunos (num_aluno,nome) VALUES (150196001,'Carlos Pina');

.Alteração

UPDATE <nome_tabela>
SET <col1**> = <**exp1>**,**
<coln>=<expn>
[**WHERE]** <condição>
Nota : sem a clausula **WHERE** todos os registos serão afectados
exp1,expn – pode ser o resultado de um query

Exp : **UPDATE alunos**
SET nome='Carlos Feijão'
WHERE num_aluno = 150196001

Exp : **UPDATE alunos**
SET data_nasc= (SELECT MIN(data_nasc) FROM alunos)
WHERE nome = 'Carlos Feijão'

(neste último exemplo o aluno de nome 'Carlos Feijão' fica com a sua data de nascimento igual à menor data de nascimento existente na tabela alunos)

.Apagar registos

DELETE FROM <nome_tabela>
[**WHERE]** <condição>

Exp : apagar todas as linhas

DELETE FROM alunos

Exp: apagar a linha correspondente ao aluno 150196001

DELETE FROM alunos
WHERE num_aluno = 150196001

QUERYS

SELECT <col1>,...,<coln>
FROM <nome_tabela1>,…,<nome_tabelak>
[WHERE] <condição>
[GROUP BY] <expressão>
[HAVING] <condição>
[ORDER BY] <expressão> **[ASC | DESC]**

Operadores :
BETWEEN, IN , IS, LIKE

Exp : Listar todos os números e nomes de alunos começados por Carlos

SELECT num_aluno,nome
FROM alunos
WHERE nome LIKE 'Carlos%'

IS deve ser usado para pesquisar nulos
SELECT num_aluno,nome
FROM alunos
WHERE data_nasc IS null

BETWEEN

```
SELECT num_aluno,nome
        FROM alunos
        WHERE num_aluno BETWEEN 150196001 AND 150196012
```

IN

```
SELECT num_aluno,nome
        FROM alunos
        WHERE num_aluno IN (150196003,150196007,150196009)
```

Funções:

ABS (n) – valor absoluto de n
MOD(m,n) – resto da divisão inteira de m por n
POWER(m,n) – devolve m levantado a n
ROUND(n,m) – devolve n arredondado a m casas decimais
SQRT(n) – raiz quadrada de n
TRUNC(n,m) – trunca n a m casas decimais
CHR(n) – char cujo código ASCII é n
LOWER(c) – devolve c com todos os chars transformados em minúsculos
UPPER(c) - devolve c com todos os chars transformados em maiúsculos
SUBSTR(c,m,n) – devolve n chars de c a partir da posição m
LENGTH(c) – comprimento da string c
TO_CHAR(d,f) – transforma a data *d* numa string com o formato *f*
TO_DATE(c,f) – converte a string *c* no formato *f* numa data
Verificar, se necessário, outras funções no manual de ORACLE. De notar que as funções variam de BD para BD e podem não ser validas para uma BD diferente do ORACLE

Exp : **SELECT UPPER(nome) FROM alunos** (nome de todos os alunos em maiúsculas)

Exemplo com cruzamento de tabelas para saber numero nome e contactos de todos os alunos :

```
SELECT num_aluno, nome,telefone
        FROM alunos A, contactos C
        WHERE C.num_aluno=A.num_aluno
```

(**A** e **C** são 'aliases' para as tabelas *alunos* e *contactos* de forma a não ter que escrever posteriormente *contactos.num_aluno* e *alunos.num_aluno*)

Funções agregadoras

MIN, MAX, COUNT(*),COUNT(expr), STDDEV,AVG, VARIANCE,SUM
Todas estas funções podem ser usadas com o DISTINCT... desta forma as linhas que para o conjunto de campos usado no SELECT sejam iguais só contam uma vez**. COUNT(expr) não conta as linhas onde expr é nulo, COUNT(*) conta todas as ocorrências incluindo duplicados e nulos.**

Exemplo usando *GROUP BY* para saber quantos alunos há de cada sexo:

SELECT sexo, COUNT(*)
FROM alunos
GROUP BY sexo

Views

CREATE OR REPLACE VIEW <viewname> [(lista colunas)]
AS <query**>**

Exp :**CREATE OR REPLACE VIEW v_contactos_aluno (nome, telefone)**
AS SELECT nome,telefone
FROM alunos A,contactos C
WHERE C.num_aluno=A.num_aluno

É como se fosse criada uma tabela virtual de nome *v_contactos_aluno* com dois campos *nome* e *telefone* e cujos registos são o resultado da query que se segue à clausula ***AS.*** Sobre esta nova tabela virtual podem ser feitas querys tais como sobre qualquer outra tabela.
Exp : **SELECT * FROM v_contactos_aluno;**
Exp : **SELECT nome,telefone**
FROM v_contactos_aluno
WHERE nome LIKE 'Rui%'

Exemplos :
Supondo as seguintes ocorrências nas tabelas ***alunos*** e ***contactos***
alunos

NUM_ALUNO	NOME	DATA_NASC	SEXO
150198001	FRANCISCO TORRES	05-10-1980	M
150198002	PEDRO SILVA	05-04-1980	M
150198003	JOÃO SILVA	03-02-1981	M
150198004	MARTA CUNHA	03-04-1980	F
150198005	SOFIA MENDES	03-07-1981	F
150198006	JORGE PALMA	01-06-1979	M

contactos

NUM_ALUNO	TELEFONE	TIPO_CONTACTO
150198001	217214433	1
150198001	919677433	2
150198002	919676423	2
150198002	217676423	1
150198003	217675523	1
150198004	217575524	1
150198006	217576621	1

1- Número de aluno, nome e contactos de todos os alunos

```
SELECT A.num_aluno numero, nome, telefone
  FROM alunos A,contactos C
  WHERE A.num_aluno=C.num_aluno
```

NUMERO	NOME	TELEFONE
150198001	FRANCISCO TORRES	217214433
150198001	FRANCISCO TORRES	919677433
150198002	PEDRO SILVA	919676423
150198002	PEDRO SILVA	217676423
150198003	JOÃO SILVA	217675523
150198004	MARTA CUNHA	217575524
150198006	JORGE PALMA	217576621

2- Número de aluno, nome e contactos de todos os alunos para alunos do sexo masculino

```
SELECT A.num_aluno numero, nome, telefone
  FROM alunos A,contactos C
  WHERE A.num_aluno=C.num_aluno
     AND A.sexo='M'
```

NUMERO	NOME	TELEFONE
150198001	FRANCISCO TORRES	217214433
150198001	FRANCISCO TORRES	919677433
150198002	PEDRO SILVA	919676423
150198002	PEDRO SILVA	217676423
150198003	JOÃO SILVA	217675523
150198006	JORGE PALMA	217576621

3- Quantos contactos tem cada aluno (listando todos os alunos mesmo os que não tem contactos)

```
SELECT A.num_aluno numero, nome, count(telefone) Quant_contactos
     FROM alunos A,contactos C
     WHERE A.num_aluno=C.num_aluno(+)
      GROUP BY A.num_aluno,nome
```

NUMERO	NOME	QUANT_CONTACTOS
150198001	FRANCISCO TORRES	2
150198002	PEDRO SILVA	2
150198003	JOÃO SILVA	1
150198004	MARTA CUNHA	1
150198005	SOFIA MENDES	0
150198006	JORGE PALMA	1

4- Quantos contactos tem cada aluno (só aparecem alunos com contactos)

```
SELECT A.num_aluno numero, nome, count(*) Quant_contactos
    FROM alunos A,contactos C
    WHERE A.num_aluno=C.num_aluno
    GROUP BY A.num_aluno,nome
```

NUMERO	NOME	QUANT_CONTACTOS
150198001	FRANCISCO TORRES	2
150198002	PEDRO SILVA	2
150198003	JOÃO SILVA	1
150198004	MARTA CUNHA	1
150198006	JORGE PALMA	1

5- Quantos contactos tem cada aluno (só para alunos com mais que um contacto)

```
SELECT A.num_aluno numero, nome, count(*) Quant_contactos
  FROM alunos A,contactos C
  WHERE A.num_aluno=C.num_aluno
  GROUP BY A.num_aluno,nome
  HAVING count(*) >1
```

NUMERO	NOME	QUANT_CONTACTOS
150198001	FRANCISCO TORRES	2
150198002	PEDRO SILVA	2

6- Quantidade de alunos nascidos em cada ano

```
SELECT to_char(data_nasc,'YYYY') Ano, count(*) Quantidade
  FROM alunos
  GROUP BY to_char(data_nasc,'YYYY')
```

ANO	QUANTIDADE
1979	1
1980	3
1981	2

Datawarehouses e Datamarts

O SI está, normalmente, optimizado para garantir a integridade dos dados à medida que a informação cresce e é modificada. Os dados sofrem modificações constantes e muitas vezes não é fácil conseguir tirar destes sistemas a informação base necessária às tomadas de decisão por parte dos gestores.
As datawarehouses foram criadas para facilitar este processo.

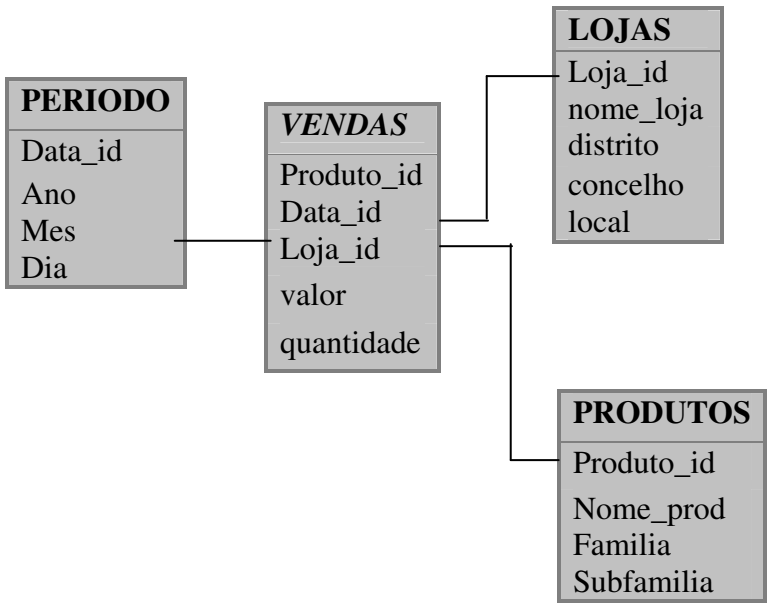
De reparar que a datawarehouse alimenta-se do SI, criando a partir deste novas estruturas optimizadas para uma fácil pesquisa e não para uma manutenção da informação.
Uma Datawarehouse encapsula toda a informação do SI enquanto que um datamart não é mais que um datawarehouse mais pequeno para resolver problemas departamentais....
Enquanto o Datawarehouse é o sistema que guarda a informação, OLAP (Online Analytic Processing) é a tecnologia que permite às aplicações clientes, aceder eficientemente aos dados...
Os dados são processados pelas ferramentas OLAP de forma a criarem uma visão, pontual no tempo, da informação. Os dados depois de processados não devem sofrer modificações (deletes ,inserts or updates) as únicas operações a serem feitas sobre os dados são operações de pesquisa.

Passos para a construção de uma data warehouse :
- Determinar os requerimentos do negócio, utilizadores e requerimentos técnicos
- Desenhar e construir a base de dados
- Extrair e carregar os dados que alimentam a Data Warehouse
- Desenhar e processar as agregações com recurso a ferramentas OLAP
- Pesquisar e manter a data warehouse e as bases de dados OLAP

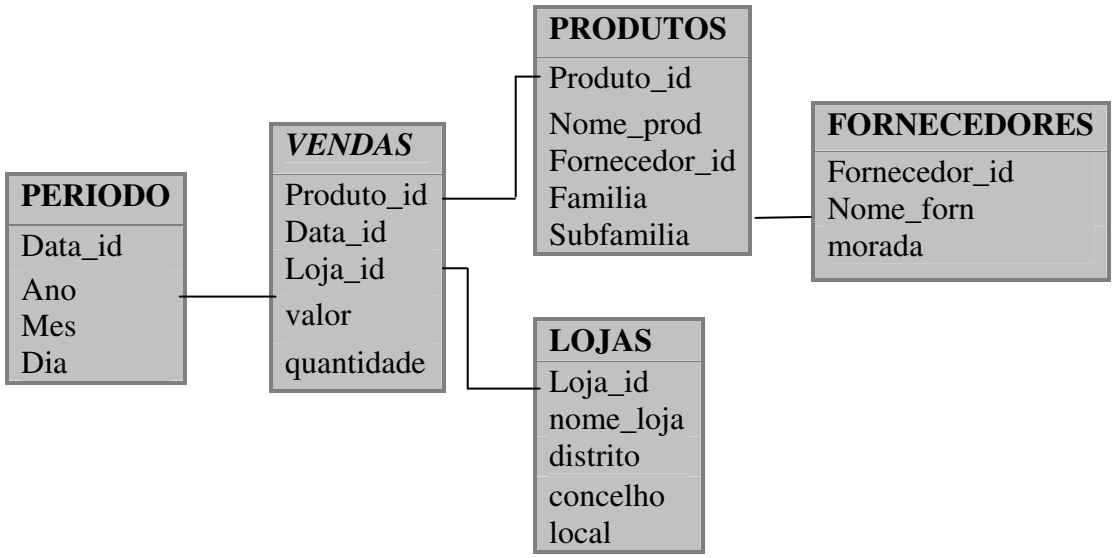
Desenho e construção da base de dados

Ao contrário de um sistema tradicional em que os dados estão organizados de uma forma normalizada, uma data warehouse organiza os dados a pensar na optimização da pesquisa e portanto de uma forma desnormalizada. Lembrar que a normalização (ver Modelo relacional) leva à construção de mais tabelas tornando, obviamente, as pesquisas mais lentas.
Os métodos de organização mais conhecidos são o ***star*** e ***snowflake***
As componentes principais deste tipo de organização são :
Fact tables:
Contem os dados que representam os acontecimentos(factos) dentro de um negócio tais como uma transação bancária ou uma venda...
A informação é estática, número de linhas muito grande (pode ir aos biliões) e representam preferencialmente dados numéricos.
Dimension tables:
Contem dados necessários para classificar os factos da ***fact table*** tais como distribuição geográfica de agencias bancárias ou categorias de produtos. Os dados não são estáticos, o número de linhas é muito menor.
Dimensões :
Categorias de informação pelas quais a Datawarehouse está organizada.

A organização em ***STAR :***
Consiste numa ***fact table*** central e uma ***dimension table*** por cada dimensão de análise.



A organização em ***SNOWFLAKE :***
É uma variação do ***STAR*** em que as ***dimension tables*** estão guardadas duma forma mais normalizada:



Bibliografia

David M. Kroenke – Database Processing Fundamentals, Design & Implementation. Prentice Hall International Editions

Dave Ensor, Ian Stevenson 1997 – ORACLE Design. O'Reilly
http://www.oreilly.com/catalog/oracledes/